Week 11 - Web Development with Flask (1/2)

Overview

This week we reviewed the Flask web development framework, and learned some of the basics that goes into creating a Flask project. For this assignment, you will be asked to create a simple web interface for a To-Do list application, using what was learned in the reading for this week.

Make sure to create a github repository for this assignment named **IS211_Assignment11**. All development should be done in this repository.

Useful Reminders

1. Read the assignment over a few times. At least twice. It always helps to have a clear picture of the overall assignment when understanding how to build a solution.
2. Think about the problem for a while, and even try writing or drawing a solution using pencil and paper or a whiteboard.

Background

For this assignment, you are to deliver a Flask application that provides a simple To Do List application. A typical To Do List application allows a user to keep a list of important tasks that they want to keep track of. In our application, a "To Do" item simply consists of:

- a short string describing the task, like "Buy Milk"
- an email address of the person responsible for the task
- a priority level, either Low, Medium or High

The application will consist of a few controllers. The first one is located at the '/' route, and will be responsible for showing the current list of To Do items in a HTML table, and showing an HTML form to submit a new To Do list item. The second controller, located at the '/submit' route, will accept data from the HTML form, add the To Do list item to a global list of items, and redirect back to the first controller. The last controller will be located at '/clear, which will be responsible for clearing the list of To Do items, and redirecting back to the first controller.

The project should be in a file called *todoapp.py*, and should be runnable by executing:

```
python todoapp.py
```

The server should then be accessible from a web browser via the URL http://localhost:5000. The following instructions will describe how the application should work.

Part I - Basic Web App Setup

First things first, create a file called todoapp.py that has a boilerplate Flask application, like the "Hello World" example, reproduced below:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

Assuming Flask is installed in your Python installation, running `python todoapp.py` should result in “Hello World” being displayed. You should base the rest of the assignment on this code

Part II - First Controller: View List of To Do Items

Our first controller that we are going to create is the controller responsible for showing the current list of To Do items. This controller should be accessible from the ‘/’ route. For this part, you are going to have to create a list that will store these To Do items. This is similar to the email list in the Flask Tutorial from this week’s reading.

> **Food for Thought:** How are you going to represent the data? We know we want to keep a list, but a list of what exactly? Think about the requirements and what we need to store about each To Do list item, and how you can best represent that data.

Next, you need to create an HTML template that displays this list of To Do items in a table (we’ll add more to this template in later parts). At the outset, this list will be empty, so for debugging purposes, you can create a ‘fake’ list of To Do items, so you can confirm that your template displays the To Do items correctly. However, make sure you clear the list before submitting a final version of your code.

Part III - Expanding the First Controller: New To Do Item Form

Once you confirm that your controller is displaying the items correctly, we need to expand on the HTML template to display a form that will be used to allow a user to submit a new To Do list item. This form should consist of the following inputs:

- a textbox input used to capture the task itself, with a name of ‘task’
- another textbox input used to capture the task’s email, with a name of ‘email’
- a drop down box allowing the user to choose from ‘Low’, ‘Medium’ or ‘High’ difficulty, with a name of ‘priority’
- a submit button with the label ‘Add To Do Item’

Remember, this form needs to submit this data to the ‘/submit’ controller, which we will build next. Until we build it, the form will not work. For this part, just focus on creating the form itself.

Part IV - Second Controller: Submitting a New Item

Now we need to create a controller for the HTML form we just created. This controller should be connected to the '/submit' URL. This controller needs to do the following:

- Receive the three input items from the form
- Do some data validation:
    - Confirm that the email input is indeed an email (HINT: think back to Week 3)
    - Confirm that the priority input value is either 'Low', 'Medium' or 'High'
    - On any error, the controller should redirect back to the '/' controller (without adding anything to the list). Optionally, you can then display an error message to indicate to the user why the submission failed.
- If there are no errors, the controller should append a new To Do item to the list
- Redirect to the main controller

At this point, you should now be able to type in a new To Do item, hit the submit button and see the new To Do item rendered in the HTML table.

Part V - Third Controller: Clear the List

The last controller is responsible for clearing the list and should be routed to the '/clear' URL. Upon being invoked, the list that stores the To Do items should be emptied and set back to an empty list. It should then redirect back to the main controller. To invoke this controller, we need to also add a new HTML form to the first controller that submits to this '/clear' controller. This form should just consist of a submit button (and for clarity, the button should display the text 'Clear')

Extra Credit I - Save the List

One of the downsides to not using a persistent data store, like a relational database, is that this list will be cleared when you restart the daemon. To alleviate this, edit your application to introduce a new form and submit button called 'Save' to the main HTML form. When invoked, the application should save the list to a file, using either the pickle module, or by saving to a file manually. Also, the application should, upon starting, check if that file exists. If it does, load the file and initialize the list as appropriate. If the file does not exist, initialize the list to be empty like before.

Extra Credit II - Delete Individual Items

It would be great if our application was able to delete individual items. Update the main HTML template to render each To Do item with a button that is labelled 'Delete'. Clicking this button should invoke a controller that will delete that To Do item from the list. There is a difficulties with doing this, however: How can we tell the controller which To Do item we need to delete? Think about how you can change the application to support this.